AfterWork

Getting Started with Apache Airflow

Learning Outcomes

- The rundown of the Apache Airflow user interface
- How to create and monitor DAGs
- The basics of using operators
- How to create dynamic workflows using Airflow

Overview

Apache Airflow is an open-source tool that was created by Airbnb designed to programmatically author, schedule, and monitor sequences of processes and tasks referred to as “workflows”. In other words, it is a workflow management system that makes sure each data pipeline task is executed in the correct order and each task gets the required resources.

For example, if you have Python scripts that should be run on schedule or run in a sequence, Apache Airflow is a convenient and reliable tool that handles that.

It offers a web UI that displays the state of currently active and past tasks, with diagnostic information about task execution. If needed, users can also manually manage the execution and state of tasks, making it automated but also allowing for human intervention.

Airflow Features

- **Easy to Use:** If you have a bit of python knowledge, you are good to go and deploy on Airflow.
- **Open Source:** It is free and open-source with a lot of active users.
- **Robust Integrations:** It will give you ready-to-use operators so that you can work with Google Cloud Platform, Amazon AWS, Microsoft Azure, etc.
- **Use Standard Python to code:** You can use python to create simple to complex workflows with complete flexibility.
- **Friendly User Interface:** You can monitor and manage your workflows. It will allow you to check the status of completed and ongoing tasks.

Airflow Use Cases

Apache Airflow is highly versatile and can be used across many many domains:

- Analytics
- Operational Work
- Data Warehousing
- Infrastructure Monitoring
- Data Exports

You can use Apache Airflow to schedule the following:

- ETL (Extract, Transform and Load) pipelines that extract data from multiple sources, and run Spark jobs or other data transformations
- Machine learning model training
- Automated generation of reports
- Backups and other DevOps tasks

Specific use case examples of Airflow include using Airflow to:

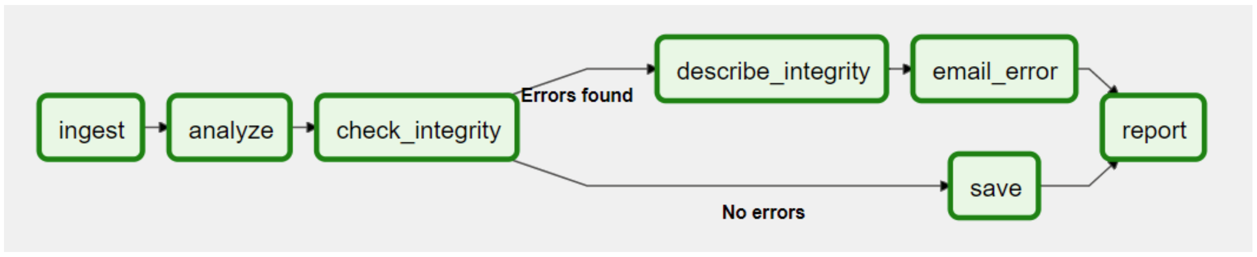
- Aggregate daily sales team updates from Salesforce to send a daily report to executives at the company.
- Organize and kick off machine learning jobs running on external Spark clusters.
- Load website/application analytics data into a data warehouse on an hourly basis.

You can find other airflow use cases here [Link].

How it Works?

DAG

The Airflow platform lets you build and run workflows, that are represented as Directed Acyclic Graphs (DAGs). A sample DAG is shown in the diagram below.



A DAG is a collection of all the tasks you want to run, organized in a way that reflects their relationships and dependencies. It is defined in a Python script, which represents the DAG Structure (tasks and their dependencies) as code.

DAG Runs
A DAG run is a physical instance of a DAG containing task instances that run for a specific `execution_date`. A DAG run is usually created by the Airflow scheduler, but it can also be created by an external trigger.

Multiple DAG runs for the same DAG can run concurrently, each with a different `execution_date`.

Tasks
A task is defined as a unit of work and can be thought of as a Python job that runs a script. It is represented by a node in your DAG and defines a unit of work in your workflow.

Task Instance
Another important concept pertaining to tasks is task instance, which runs a single task and has a state such as `running`, `success`, `failed`, `skipped`, or up for `retry`, etc.

Operators
In Airflow, we use operators to define tasks. Once an operator is instantiated within a given DAG, it is referred to as a task of the DAG.

There are three main types of operators:

1. **Action Operators:** Operators that perform an action or tell another system to perform an action i.e. PythonOperator or BashOperator.
2. **Transfer Operators:** Operators that move data from one system to another i.e. S3ToRedshiftOperator.
3. **Sensor Operators:** Operators that keep running until a specific criterion is met. Their purpose is to wait on some external or internal trigger. They are commonly used to trigger some or all of the DAG, based on the occurrence of some external event. Some common types of sensors include:
    - `ExternalTaskSensor:` waits on another task (in a different DAG) to complete execution.
    - `HivePartitionSensor`: waits for a specific value of partition of hive table to get created.
    - `S3KeySensor`: S3 Key sensors are used to wait for a specific file or directory to be available on an S3 bucket.

Below are a few other operators with explanations of what they do:

- `BashOperator` – used to execute bash commands on the machine it runs on.
- `PythonOperator` – takes any python function as an input and calls the same.
- `EmailOperator` – sends emails using SMTP server configured.
- `SimpleHttpOperator` – makes an HTTP request that can be used to trigger actions on a remote system.

- `MySqlOperator`, `SqliteOperator`, `PostgresOperator`, `MsSqlOperator`, `OracleOperator`, `JdbcOperator`, etc. – used to run SQL commands.
- `Qubole Operator` – allows users to run and get results from Presto, Hive, Hadoop, Spark Commands, Zeppelin Notebooks, Jupyter Notebooks, Data Import / Export Jobs on the configured Qubole account.

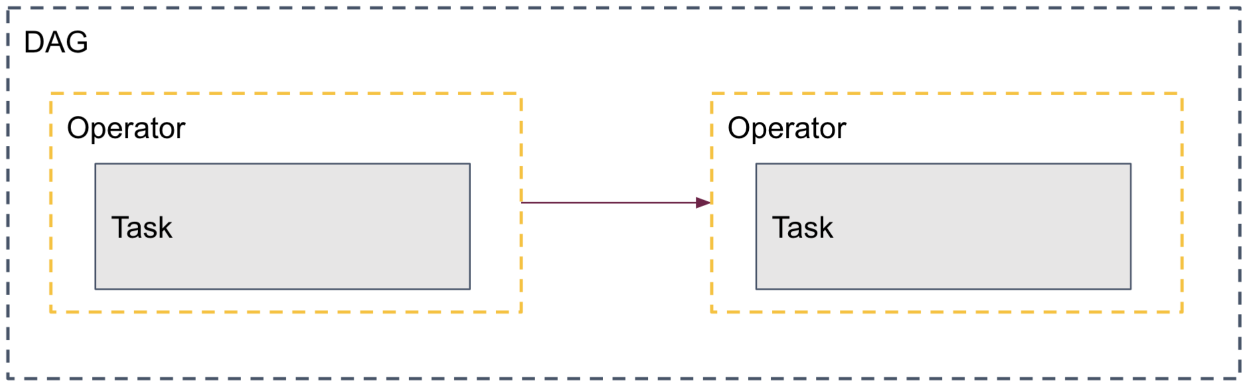
Operator Code Examples:

```
t1 = BashOperator(task_id='print_date',
    bash_command='date,
    dag=dag)

def print_context(ds, **kwargs):
    pprint(kwargs)
    print(ds)
    return 'Whatever you return gets printed in the logs'

run_this = PythonOperator(
    task_id='print_the_context',
    provide_context=True,
    python_callable=print_context,
    dag=dag,
)
```

At a high level, the combined system of DAGs, operators, and tasks looks like this:



Further Understanding Airflow DAGs

Airflow DAGs are Python scripts with 5 main sections

1. **Imports**

   Code Example

   ```
   from airflow import DAG
   from airflow.models import Variable

   from airflow.contrib.bigquery_operator import BigQueryOperator
   from airflow.contrib.operators.mlengine_operator import
   MLEngineVersionOperator
   from airflow.operators.dummy_operator import DummyOperator
   from airflow.contrib.operators.pubsub_operator import
   PubSubPublishOperator
   from airflow.operators.python_operator import PythonOperator
   from airflow.ultils.trigger_rule import TriggerRule
   ```

2. **Arguments**

   Define default and DAG-specific arguments. We can define a dictionary of default parameters to be used when creating tasks

   ```
   DEFAULT_ARGS = {
         'owner' : 'Valentine Mwangi',
         'depends_on_past': False,
         'start_date': datetime.datetime(2019, 12, 1),
         'email': ['valentine@example.com'],
         'email_on_failure': False,
         'email_on_retry': False,
         'retries': 1,
         'retry_delay': datetime.timedelta(minutes=5)
   }
   ```

3. **Instantiation**

   Name the DAG and set the `dag_id`, which serves as a unique parameter for your DAG.

   You also configure the DAG schedule and other DAG settings via the default argument dictionary that you just defined.

```
with DAG(
      'chicago_taxi_dag',
      catchup = False,
      default_args = DEFAULT_ARGS,
      schedule_interval = '@monthly') as dag:
```

You can also specify a DAG by instantiating an object of the `airflow.models.dag.DAG` class as follows:

```
dag = DAG('Example1',
           schedule_interval='@once',
           start_date=days_ago(1),)
```

The DAG above would appear in the webserver UI as “Example1” and will run once.

4. **Tasks**

   Airflow provides operators for performing many common tasks, including:
   - **BashOperator**: Executes a bash command.
   - **PythonOperator**: Calls an arbitrary Python function

   In the case of Google Cloud, Airflow provides operators for tasks i.e. BigQuery Operators, Cloud Storage Operators, Dataflow operators, Cloud build operators etc.

```
bq_train_data_op = BigQueryOperator(
      task_id = "bq_train_data_task",
      sql = sql_train,
      destination_dataset_table = ...,
      write_disposition = "WRITE_TRUNCATE",
      use_legacy_sql = False,
      dag = dag
)
```

   Simply put, operators define the work. When instantiated in Python code, an operator is called a task. A task is an instantiation of an operator and can be thought of as a unit of work.

5. **Dependencies**

   Airflow excels at defining complex relationships between different tasks.

   Operator relationships are set with the `set_upstream()` and `set_downstream()` methods. This can also be done with the Python bitshift operators >> and <<.

The following four statements are all functionally equivalent:

```
op1 >> op2
op2 << op1
op1.set_downstream(op2)
op2.set_upstream(op1)
```

The bitshift operators can also be used with lists: for example:

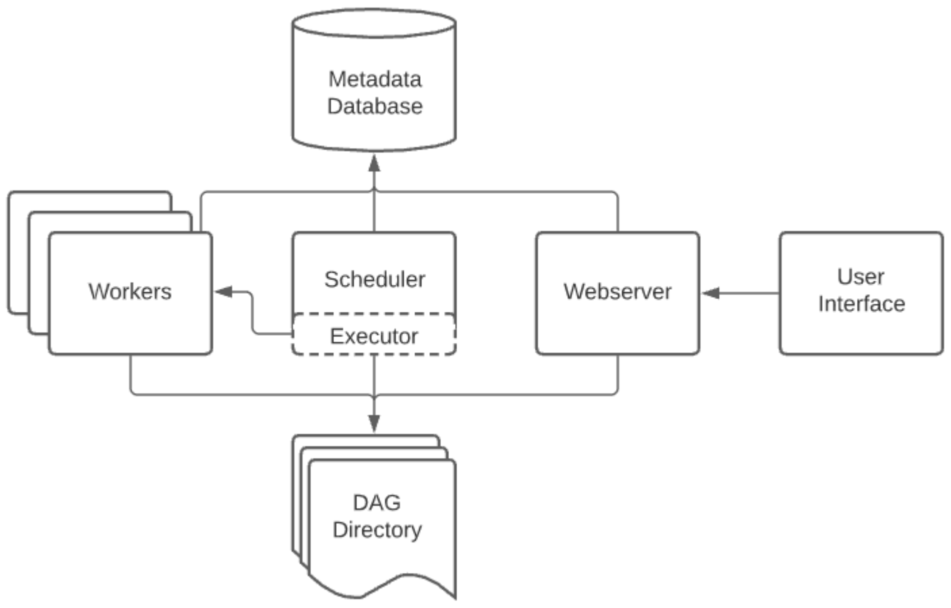
```
op1 >> [op2, op3] >> op4
```

is equivalent to:

```
op1 >> op2 >> op4
op1 >> op3 >> op4
```

Airflow Architecture

The following diagram shows typical components of an Airflow architecture.



The Scheduler

- The scheduler is responsible for monitoring all DAGs and the tasks within them. When dependencies for a task are met, the scheduler triggers the task. Under the hood, the scheduler periodically inspects active tasks to trigger.

The Web Server

- The web server is Airflow's UI. It displays the status of the jobs and allows the user to interact with the databases as well as read log files from a remote file store, such as S3, Google Cloud Storage, Azure blobs, etc.

The Executor

- This is the mechanism that gets the tasks done i.e. executes the logic of tasks. This component handles the coordination and execution of different tasks across multiple DAGs. There are many types of Executors in Apache Airflow, such as the SequentialExecutor, LocalExecutor, CeleryExecutor, DaskExecutor, and others.

Metadata Database

- Airflow stores the status of all the tasks in a database and does all read/write operations of a workflow from here.

How Tasks Are Executed

1. Airflow parses all the DAGs in the background at a specific period. The default period is set using the `processor_poll_interval` config, which is, by default, equal to one second.

2. Once a DAG file is parsed, DAG runs are created based on the scheduling parameters. Task instances are instantiated for tasks that need to be executed, and their status is set to SCHEDULED in the metadata database. Since the parsing takes place periodically, any top-level code, i.e., code written in global scope in a DAG file, will execute when the scheduler parses it. This slows down the scheduler's DAG parsing, resulting in increased usage of memory and CPU. Therefore, caution is recommended when writing code in a global scope.

3. The scheduler is responsible for querying the database, retrieving the tasks in the SCHEDULED state, and distributing them to the executors. The state for the task is changed to QUEUED.

4. The QUEUED tasks are drained from the queue by the workers and executed. The task status is changed to RUNNING.

5. When a task finishes, the worker running it marks it either failed or finished. The scheduler then updates the final status in the metadata database.

Other Airflow Concepts

Hooks

- Hooks are Airflow's way of interfacing with third-party systems. They allow you to connect to external APIs and databases like Hive, S3, GCS, MySQL, Postgres, etc. They act as building blocks for operators. Secure information such as authentication credentials are kept out of hooks - that information is stored via Airflow connections in the encrypted metadata DB that lives under your Airflow instance.

Providers

- Providers are community-maintained packages that include all of the core Operators and Hooks for a given service (e.g. Amazon, Google, Salesforce, etc.). As part of Airflow 2.0, these packages are delivered multiple, separate but connected packages and can be directly installed in an Airflow environment.

Plugins

- Airflow plugins represent a combination of Hooks and Operators that can be used to accomplish a certain task, such as transferring data from Salesforce to Redshift.

Connections

- Connections are where Airflow stores information that allows you to connect to external systems, such as authentication credentials or API tokens. This is managed directly from the UI and the actual information is encrypted and stored as metadata in Airflow's underlying Postgres or MySQL database.

Resources

1. Apache Airflow Documentation [Link]
2. Data Pipelines by Apache Airflow [Link]
3. Data Engineering 101 - Getting started with Apache Airflow [Link]
4. The Ultimate Guide to Apache Airflow [Link]